Macros and K Programs

12. Macros and K Programs 12-1
12.1 Macros 12-1
12.2 K Programs 12-2
Initiate a K program 12-2
Automatic Execution 12-3
Error Messages 12-3
Read and Record a K Program 12-3
Format and Syntax 12-4
12.3 The K Editor 12-4
Keystrokes 12-5
Commands 12-6
Comments 12-7
K Editor Keystrokes 12-7
12.4 K Programming Language 12-10
K Data Types 12-10
K Variables 12-11
Arrays 12-12
Global and Local Variables 12-12
System Variables 12-13
The List of System Variables 12-14
Expressions 12-16
K Commands by Category 12-16
Loops and Conditionals 12-18
Subroutines 12-20
K Program Parameters 12-21
Returning from a K Subroutine 12-23
Exiting from a K Program 12-24
Data Type Conversions 12-24
String Operators 12-25
Cursor Ouput Commands 12-27
Control the Keyboard and K Playback 12-28
Serial Input and Output 12-31

12. Macros and K Programs

12.1 Macros

As you operate a K-Series character generator, you'll sometimes repeat the same keystroke sequences. This can be tiresome and may lead to errors. Instead, press **Learn** to create keyboard macros. A ***keyboard macro*** is a user-defined Do Command that automatically types a keystroke sequence.

To create a keyboard macro:

1. Press **Learn**. The monitor says `LEARNING`.
2. Type the keystrokes you want the K-Series character generator to remember. The keys will do what they normally do.
3. Press **Learn** again to stop learning. The monitor prompts for a name.
4. Type the name of the macro and press **Enter**. The K-Series character generator automatically records the macro.

To play back a keyboard macro:

1. Press **Do**.
2. Type the name of the macro.
3. Press **Enter**.

In effect, you have created your own Do command!

IMPORTANT TRICK: If you have a frequently-used macro, give it the name `ALTa` when you record it. The program does not differentiate between upper case and lower case, so ALTA is the same as ALTa. To play it back, just press `Alt` followed by **a**. You don't need to press **Do** or **Enter**. That way, you can play back your macro by pressing just two keys. You can use all of the letters of the alphabet to make macros `ALTa` through `ALTz`, (By the way, you can also type **Do** `ALTa` **Enter** for these.)

Let's imagine a situation where keyboard macros might be useful. Say you've already recorded 30 pages of text for some closing credits. The text is all in font 232, and all the pages are left-justified. Now the director comes in. "That's great," she says. "But... could you do the whole thing over with a smaller font, and all of the text centered?" Your first impulse is to panic; you're on the air in seven minutes! Don't worry. Here's what you do.

Assign the new, small font to Selector 1. Read in the first message of the series. Then press the following keys:

Learn Text P Centr P Alt Rec Learn

Record the macro at **ALTc**. You have just fixed and rerecorded the first page of text. To fix the next page, read it in and type **Alt c**. And so on. You can even make the Read part of the macro.

While learning, the K-Series character generator keeps track of all functions that are executed by building a list. Learn builds the list into the ***K Edit buffer***. To see the list, press **K-Edit**. With the K Editor, the K programs can be edited and combined with loops, variables, if-then-else, and other programming tools to create powerful, customized functions. We explain editing functions later. Press **Compose** to leave the K Editor.

Many systems call these lists of functions "histories" (usually a repeat of menu choices) or "macros" (usually a list of keystrokes). These lists are known as K Programs.

12.2 K Programs

A ***K program*** is a list of keystrokes and commands.

The K program runs through the list. Commands are executed. Keystrokes are interpreted as if someone were typing.

You can't start a K program in the middle; the program always begins at line 1. To stop a program before it finishes, press **Quit**.

Initiate a K program

K programs are commands. To initiate the K program by name from the Do command line, follow the procedures for entering Do commands. See "Introduction to Do Commands" in the User's Guide.

Frequently used K programs can be assigned to ***softkeys***. A K program with the name "ALT" followed by a character "a" through "z" is invoked at the **Alt** function of the typewriter key.

K programs can be initiated directly from the K Editor by pressing **Enter Enter**. K programs don't have to be recorded to play from the K Editor.

Automatic Execution

Immediately after reboot, the K-Series character generator automatically executes any K program name `INIT` on the program drive.

Error Messages

When a K program encounters an error, a prompt reports the problem command and parameter. If the program is in the K Edit buffer, the cursor is placed at the line of the error.

If the program isn't loaded in the K Edit buffer when an error is encountered, enter K Edit mode. The keystroke **Sel-8** reads the K program and places the cursor at the line of the error.

The keystroke **Alt Sel-8** places the cursor at the line of the last error.

Read and Record a K Program

By default, K programs can be read only in the K Editer. If you try to read a K program outside the K Editor, the status monitor prompts

```
K message ignored.  Not in K Editor
```

K programs are recorded like any other message. From the K Editor

Record *name* **Enter**

(K programs can also be recorded immediately after being learned, without entering the K Editor.)

Since K programs are commands, record them with a single word. Don't use any punctuation, other than an underscore or a period, in the name. A K program can't begin with a number.

Don't give a K program the same name as a system command. If you do, the system command (not the K program) will be invoked.

Use **Confg X** to configure the character generator to read a K program from the Directory. Press **Read-Next** to read a K program outside the K Editor (if it is next in the Directory list).

Format and Syntax

This section includes syntax rules and guidance for formatting K programs.

There is usually a single command per line. Commands within K programs appear as they would on the Do command line.

If the command and all its parameters do not fit on a single line, a backslash (\) indicates that the command continues on the next line:

```
combine firstmessage="aardvark" lastmesage="zebra" \
  newname="animals_anim"
```

Separate multiple commands along a line with semi-colons(;):

```
type "K-Series 20K or 30K"; <Centr>c
```

Multiple keystrokes can appear continuously. No special characters are necessary to separate them or to continue them along multiple lines.

```
<Sel-1>Grass Valley Group<Centr>c
```

Use tabs and blanks to make your K program legible. For example, indenting commands within a condition block or a loop greatly improves the legibility of your program. The K interpreter ignores this formatting.

12.3 The K Editor

Press **K** to enter the K editor. The ***K editor*** is a word processor for creating and modifying K programs. The monitor displays the program you are currently editing.

The K-edit language is the language used in the K Editor. When the character generator is configured for a language other than English, for example, French or German, use the English keyboard layout for accessing all commands. To type, for example, a string in French in a K program, you can switch the K-edit language to French. See **Confg K** in "Login and Logout" in the User's Guide.

Use the cursor keys to move about the screen. Use **Insrt**, **Del**, and **Backspace** to add or delete text. Use **Shift-Row** and **Shift-Char** to scroll through multiple pages. **Alt Newline** duplicates the cursor row. **Alt Del** deletes the cursor row.

Keystrokes

To enter typewriter keystrokes into a K program, first press **K**. The bottom of the monitor tells you you're in `<<<Keystroke>>>` mode. Then press as many typewriter keys as you like to enter into your program. Press **Spacebar** to leave keystroke mode.

To enter a keystroke into a K program, first press **K**, then press the key. For example, **K Home** puts the **Home** key into the K program, and *does not home the cursor*. The K editor leaves keystroke mode automatically, after you press **Home**.

The keystroke is indicated on the status monitor in brackets `<Home>`. *Do not type out the keystroke's name letter by letter.* If you position the cursor over the keystroke, the entire keystroke is highlighted.

Learn is another way to enter keystrokes into a K program. While you're in the K editor, press **Learn**. You'll be back in Compose mode, and the monitor reports that you are `LEARNING`. Any key you press not only does what it normally does, but is also entered into your K program. To get back to the K editor and stop learning, press **Learn** again.

Let's try an example. Press **K** to enter the K editor. Press **Erase-Text** twice to clear any previous K program that might have been left around. Now press **Learn**. You're back in Compose mode, learning.

Type `Hello` **Newline** `K-Series.` Center it with **Centr M**. Okay, that's enough—press **Learn** to stop learning. The K editor shows you what it has learned:

```
Hello <Newline> K-Series <Centr> m
```

Why are `Hello` and `K-Series` underlined? In a K program, letters and numbers represent either keystrokes or K variables and commands. To differentiate, the K editor underlines any characters that are meant to be typed literally by the K-Series character generator. If the characters are not underlined, then they are K commands or variables. For example:

`select 1 0`	is the command to select drive 1 for messages, drive 0 for fonts.
`select 1 0`	types the characters s-e-l-e-c-t-space-1-space-0 on the monitor, using the current selector.

<Newline> and <Centr> are the K editor's version of **Newline** and **Centr**. Move the K editor cursor to <Newline>. The cursor highlights the entire key, since it represents a single keystroke.

Commands

Enter commands into K programs by typing them as you would a Do command. Don't press **K** first. For example:

```
select 0 1

copy  5  1  shoe
```

In the K editor, commands are not underlined because the letters and numbers are not "typed"; the command is executed directly by the K interpreter.

To put more than one command on a single line of a program, separate the commands with semicolons:

```
select 0 1; fscale 60001 132 32; select 1 1
```

To make a command extend beyond one line, use a backslash:

```
fscale 60001 132 32 -all hscale=80 \
  comment="Clarendon 32 no edge" \
  filter=narrow
```

The commands, the parameters, and the syntax can be a lot to remember. Luckily you don't have to. You can "learn" anything you execute directly into the K Editor. See the section "Keystrokes," above.

Comments

To add a ***comment*** to a K program, use a pound sign (#):

```
# this statement does nothing
```

Anything to the right of a pound sign on that line is ignored when a K program is executed. Comments can be placed on the same line as commands, as long as the comment is to the right of the command.

K Editor Keystrokes

These work in the K Editor:

Sel-1	Save current line for processing: Mark 1.
Alt Sel-1	Display Mark 1.
Sel-2	Save current line for processing: Mark 2.
Alt Sel-2	Display Mark 2.
Sel-3	Delete block and clear marks.
Sel-4	Copy block into save buffer.
Alt Sel-4	Clear save buffer, then copy block into save buffer.
Sel-5	Cut block into save buffer.
Alt Sel-5	Clear buffer, then cut block into save buffer.
Sel-6	Put contents of save buffer into current K program.
Alt Sel-6	Put contents of save buffer into current K program, then clear save buffer.

Sel-7	Clear save buffer.
Compose	Leave the K Editor.
Help	Help for the K Editor.
Enter Enter	Play the current K program.
Do	Copy **Do** command line into the current K program.
Alt Do	Copy the currrent line in the K program into the **Do** command line.
Read	Read a K program, replacing the current K program.
Alt Read	Read a K program, appending it to the current K program at the cursor location.
Rec	Record the K program.
Alt Rec	Re-record the current K program.
K-Edit	Enter keystroke mode (the next key is a keystroke to be typed into the K program).
Erase-Text	Erase the K program.
Tab	Tab to the next tab stop (4 spaces each stop).
Insrt	Toggle ***insert/typeover*** modes.
Alt Insrt	Insert a row.
Newline	In insert mode, add new line; otherwise position cursor at the beginning of the next line.
Alt Newline	Copy the cursor row and insert it one row below.
Learn	Leave the K Editor and begin learning keystrokes (when you return to the K Editor, all learned commands and keystrokes are inserted at the cursor location).

Home	Move the cursor to the first character of the first page of the program.
Alt Home	Move the cursor to the last line of the program.
Del	Delete the cursor character. To join two rows, move the cursor past the last word of the first row and press **Del**.
Alt Del	Delete the cursor row. All rows below move up.
Backspace	Delete the character to the left of the cursor. If the cursor is on the first character of the row, move the cursor row up to the end of the row above.
Shift-Row ↓	Display the next page of the program.
Shift-Row ↑	Display the previous page of the program.
↑	Move the cursor up one row.
↓	Move the cursor down one row.
Alt ↑	Move the cursor to the top row of the current page.
Alt ↓	Move the cursor to the bottom row of the current page.
Kern/Xchg ↑	Exchange the cursor row with the one above.
Kern/Xchg ↓	Exchange the cursor row with the one below.
Shift-Char ↓	Scroll through the program down one row.
Shift-Char ↑	Scroll through the program up one row.
Alt ←	Move the cursor to the first character of the row.
Alt →	Move the cursor to the last character of the row.

12.4 K Programming Language

K is the K-Series' own programming language. Using K, you can tailor the character generator to fit almost any situation. K is especially well-suited to real-time applications where you must respond to changing conditions on the fly. K programs can "play" a game show, report sports results, assemble a series of images and create an animation from them, control another computer, be controlled *by* another computer. And so on.

K is a true programming language. It combines many of the best features of C, BASIC and K's predecessor, KPL. K supports subscripted variables, floating point and integer arithmetic, variables, arrays, if-then-else and loops, conditionals, and subroutines.

K Data Types

A K program can have four types of data:

Integer, Real, String, Keystroke

An ***integer*** is a whole number value—one with no fractions. Integers can be positive or negative.

A ***real*** is any number value: whole, fractional, or a mix of the two. Real numbers always contain a decimal point and are expressed as ***floating point*** numbers. When dividing, real numbers provide more precision than integers.

A ***string*** is a set of characters, ie., a word or sentence. When defining a string surround it with quotation marks if it includes any spaces or non-alphanumeric characters. Message names and sometimes command names are specified as strings.

A ***keystroke*** is a character generator keystroke. To put a keystroke into a K program while in the K Editor, press K followed by the key. Do not type the key's name letter by letter.

All types of data can be assigned to variables, can be passed to commands as parameters, or can be returned from commands as variables.

K Variables

Variables in K are similar to variables in algebra. Within K programs and commands, values can be represented by a variable name. The variable's value can change (but a variable can only represent one value at a time). Any type of K data (keystrokes, integers, real numbers, and strings) can be assigned to a variable.

A K variable name can have up to 16 characters, with the first character always being a dollar sign ($). It may include upper and lower case letters and numbers, but not imbedded punctuation (exception: underline is permitted). The character immediately after the $ cannot be a number. Here are some examples of valid K variable names:

```
$i
$height
$last_place_team
$MiddleInitial
$horse12
$state[50]
```

HINT: Use names that are meaningful to the value being represented. It helps to make your programs easier to read and to understand.

The K-Series character generator creates a K variable when you assign a value to it. The type of data that the variable represents determines the type of variable that is created. You don't need to declare K variables explicitly. To assign a variable, use the form

value = expression.

Here are examples of assignment statements for each variable type:

ASSIGNMENT	*TYPE OF VARIABLE*
$i = 1	integer
$string = blah	string
$fred = "hello world"	string
$position = 1.234	real number (floating point)
$key = <Home>	keystroke

If you try to use a variable before a value is assigned to it, the character generator stops executing the program and displays the error message:

```
variable not found
```

Arrays

K variables may be followed by one or more subscripts. The subscript must be an integer (or an expression that evaluates to an integer) and is indicated by block brackets. Some examples:

```
$horse[12]
$state[$i]
$a[3 * ($b + $c)]
```

The character generator stores subscripted variables the same way it stores simple variables, and does not allocate memory for arrays. Thus the three K variables `$a[1]`, `$a[5]`, and `$a[9999]` use only three K variable slots, not ten thousand.

Global and Local Variables

Any K variables that you use in a program are local to that program. ***Local variables*** are allocated temporarily during the execution of a K program, then automatically deleted when the program terminates. Program B can not reference Program A's local variables, even if Program B is a subroutine called by Program A.

In order for multiple K programs to share variables, the variables must be declared global, using the `global` command:

`global` *$variable1* [*$variable2* *$variable3*...]

Once they have been declared, ***global variables*** stay active until you reboot the character generator, even if no K program is currently running. Any K program may reference any global variable.

The amount of memory reserved for global variables is not infinite, and it is possible to fill it up if you use a large number of globals. However, you can release any global variables that are no longer needed, using this command:

`free_globals` [*$variable1* *$variable2* *$variable3*...]

If no variable names are specified, `free_globals` releases all global variables. Use this command to avoid other K program variables from interfering with the current K program.

System Variables

A ***system variable*** is a predefined global variable that is always available to commands and K programs. The first character of these variables is always an "at sign," so we call them ***at sign variables.*** System variables can be integers, real numbers, or strings. We list the system variables below.

Many system variables describe some aspect of the K-Series character generator. During normal operation, they are updated so that they show the current system status. Some examples:

The system variable `@mdrive` is the value of the current message drive. If you change your message drive, `@mdrive` also changes.

`@selector` and `@selkey` (an integer value, one to eight, representing the current selector) change values as you change selectors.

`@dots` and `@lines` are the number of horizontal and vertical pixels of an image. It requires a change to a different video standard (and a reboot) for these values to reflect the new status. In this regard, these system variables act as constants.

The List of System Variables

NAME	TYPE	EXAMPLE	DESCRIPTION
`@version`	string	"2.0"	Program version
`@pgmdate`	string	"6 Feb, 1992"	Program date
`@date`	string	"Sat 6 May, 1992 12:53:54"	Current date and time
`@mdrive`	string	"1"	Current message drive
`@fdrive`	string	"2"	Current font drive
`@pdrive`	string	"1"	Drive program was booted from
`@mode`	keystroke	<Compose>	Current mode: <Compose>, <Do>, <Pal>, etc.
`@dots`	integer	670	Number of pixels on one scan line
`@lines`	integer	486	Number of visible scan lines
`@leftsafe`	integer	76	Pixel (dot) # of left side of safe title area
`@rightsafe`	integer	603	Pixel (dot) # of right side of safe title area
`@topsafe`	integer	48	Scan line number of top of safe title area
`@botsafe`	integer	437	Scan line number of bottom of safe title area
`@wordwrap`	integer	1	1 if wordwrap is enabled, else 0
`@insert`	integer	1	1 if insert is enabled, else 0
`@interleave`	integer	2	Iomega disk, interleave value (1,2,4)
`@oddfield`	integer	1	1 if effects to start on odd field, 0 if even field
`@ticker`	integer		A counter which advances at start of each field
`@selector`	integer	1	Current selector number (1-8)
`@selkey`	keystroke	<Sel1>	Current selector key
`@selfont`	integer	1024	Font number of current selector
`@cursor`	integer	1	1 if cursor on is enabled, 0 if off

`@cursorfont`	integer	`1024`	Font number of character under text cursor
`@cursorchar`	integer	`41`	ASCII code of character under text cursor, `-1` if there is no character
`@cursorword`	string	`"pretty"`	Character string of word under the text cursor
`@currow`	integer	`300`	Scan line number that cursor is on
`@curdot`	integer	`400`	Dot number that cursor is on
`@hue`	integer	`240`	Current color's hue
`@saturation`	integer	`999`	Current color's saturation
`@value`	integer	`999`	Current color's luminance
`@red`	integer	`255`	Current color's red value
`@green`	integer	`255`	Current color's green value
`@blue`	integer	`255`	Current color's blue value
`@opacity`	integer	`100`	Current color's opacity
`@horizpaint`	integer	`0`	1 if "painting" scan lines, else 0
`@horizline`	integer	`150`	scan line number that "paint" cursor is on
`@kbdport`	string	`"A"`	Port ID of the current keyboard's serial port
`@infield`	string	`"foo"`	The character string in current menu entry field
`@key_avail`	integer	`1`	1 if a keystroke is available from keyboard, 0 if not

Expressions

Arithmetic and logical expressions may be used in assignment statements. K uses a subset of C's operators and precedence rules, with a few extensions. The following table summarizes the operator precedence rules, in order of decreasing precedence:

`[]`	Subscript
`()`	Parentheses
`! ~ -`	Logical not, one's complement, unary minus
`* / %`	Multiply, divide, remainder
`+ -`	Plus, minus
`<< >>`	Shift left, shift right
`%%`	Concatenate strings
`< <= > >=`	Comparison operators
`== !=`	Equality, inequality
`&`	Bitwise AND
`^`	Bitwise exclusive OR
`|`	Bitwise OR
`&&`	Logical AND
`||`	Logical OR

There are no ++ or += operators.

Use (nested) parentheses to remove ambiguity in arithmetic expressions:

```
$f = 32 + (1.4 * $c)

$inc = (($i+1) * ($j+1)) / 100
```

K Commands by Category

Here is a brief description of each of the K commands, arranged by category:

Loop and Conditionals

`loop`	Repeat a section of a program.
`end`	Terminate an `if`, `loop`, or `while`.
`if`	If condition is true, execute a section of a program,
`else`	otherwise execute a different section.
`while`	While condition remains true, continue executing a section of a program.
`continue`	Return to the top of a `while` or `loop`.
`break`	Break out of `while` or `loop`.

Variables, Subroutines and Parameters

`global`	Define a global variable.
`free_globals`	Remove a global variable.
`keyword`	Define a K subroutine keyword parameter.
`return`	Exit from a K subroutine.
`exit`	Terminate a K program.
`exit_option`	Select options for `exit`.

Cursor Output

`cursor`	Position the cursor.
`newline`	Advance cursor to start of next line.
`cursor_on`	Turn on cursor.
`cursor_off`	Turn off cursor.
`print`	Display cursor output text on monitor.
`message`	Display cursor output text on message line of monitor.
`center`	Center cursor output text on current line of monitor.
`clear_line`	Clear cursor output line of monitor.
`clear`	Clear the cursor output from screen.
`status`	Restrict cursor output prompting.

Serial Input and Output

`getport`	Return port number of a specific application.
`input`	Input from one of the serial ports.
`output`	Output to one of the serial ports.
`clearinput`	Clear input buffer for a port.
`testinput`	Test a serial port.

Data Type Conversions

`asc2num`	Turn a character string into an integer or real number.
`num2asc`	Turn an integer or real number into a character string.
`str2num`	Turn a string into the numeric value of that string.
`num2str`	Turn a binary number into a character string.
`integer`	Return integer part of a real number.

String Operators

`substr`	Return a substring within a character string.
`index`	Find a string within another string.
`length`	Return the length of a string.

Keyboard control and K playback

`kdrive`	Specify the drive where K programs live.
`getkey`	Input a single keystroke from the keyboard.
`type`	Type a key.
`getnum`	Input a number from the keyboard.
`getstring`	Input a string from the keyboard.
`user`	Suspend K playback, return control to the user.
`wait`	Suspend K playback, wait a specified time.
`trigger`	Wait for GPI trigger.
`user_func`	Suspend K playback, until effect or menu is done.
`k_func`	Don't wait for the end of the effect, continue executing K program.

Loops and Conditionals

A ***loop*** is a part of a program that is repeated more than once in a program. Provisions for forming a loop are essential to all programming languages.

Any function can be repeated, over and over within a loop. In K, there are a number of ways to define the block of commands and keystrokes to be repeated.

Loops can be defined to repeat a certain number of times or some sort of test can determine when the loop should be broken. An endless loop repeats forever.

```
loop [count=]
  keystrokes
  commands
end
```

Repeat the *keystrokes* and *commands* between the `loop` and its corresponding `end` the number of times specified. If no *count* is supplied, loop forever, unless a `break` statement is encountered. (Press **Quit** to break out of an endless loop and stop the program.) Loops may be nested. The following program types "hello" five times, once per line:

```
loop 5
    hello <New-Line>
end
```

`while` *expression*
keystrokes
commands
`end`

Repeat the *keystrokes* and *commands* between the `while` and its corresponding `end` for as long as the `while` expression is true. The `while` expression is reevaluated each time the loop starts over.

The following program is equivalent to the program above:

```
$a = 1
while $a <= 5
   hello <New-Line>
   $a = $a + 1
end
```

`if` *expression*
keystrokes
commands
`[else`
keystrokes
commands`]`
`end`

Conditional if-then-else. Evaluate *expression*; if true, execute `if` portion; if false, execute `else` portion. The `else` portion is optional. Remember, to put more than one K command on a single line of a program, the commands must be separated by semi-colons:

```
if $a==0; <Enter>; else; <Tab>; end;
```

`continue`

Restart the current loop. The current loop is the nearest `while` or `loop` command, directly above the `continue`.

`break`

Exit from the current `while` or `loop` by skipping down past the nearest `end`.

The following program (which doesn't really do anything) shows how to use `continue` and `break`:

```
while $a <= 5
  if $b == 2        # this is a comment
    continue        # goes back to "while"
  end
  if $b == 3
    break           # exit from "while" loop
  end
end
```

HINT: Indent lines that are in a conditional block or loop. K doesn't require this type of formatting; the programs run just as well without it, but indenting makes the programs much easier to read. It's a good habit.

Subroutines

A K program may call another K program as a subroutine. For example, say you've already recorded a K program called `rubble`:

```
# the rubble family
<Newline> Barney
<Newline> Betty
<Newline> BamBam
```

and another K program, called `flintstone`:

```
# the flintstone family
<Newline> Fred
<Newline> Wilma
<Newline> Pebbles
```

You can call these two K programs as subroutines from a third K program, `bedrock`:

```
# names of Bedrock inhabitants
rubble
flintstone
```

As long as the program `rubble` can be found on the disk, it can be used as a subroutine. But it is often preferable not to rely on other K programs being on the disk.

If a K program selects a new message drive, it can get confused about where to find subroutines. Also, you may get confused when moving K programs and forget to move the subroutine files. In general it is preferable to make subroutines internal to the K program.

K Program Parameters

Like Do commands, K programs can have positional, flag and keyword parameters. This is true both for K programs that an operator starts with the **Do** button, and for K programs that are called as subroutines from other K programs.

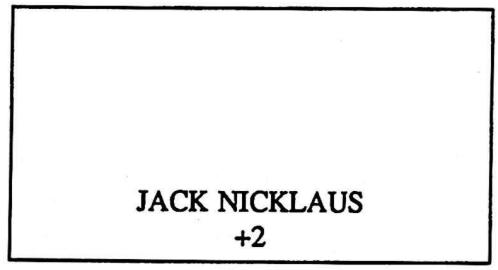
Positional parameters are defined implicitly, and automatically assigned to the pseudo-variables `$1`, `$2`, and so on. (The pseudo-variable `$0` contains a count of how many parameters have actually been passed.) For example, here's a program called `golf` that types two lines of text as a lower third:

```
# type 2-line lower third
# using selectors 1 and 2
<Erase-Text>
<Sel-1>; type $1; <Newline>
<Sel-2>; type $2; <Center> 1
```

`$1` and `$2` refer to the first two positional parameters, when the `golf` command is invoked. To run the `golf` program, go to the character generator and type

Do `golf "JACK NICKLAUS" "+2"` **Enter.**

The monitor displays:



Keyword Parameters must be declared at the start of a K program, using the command

`keyword` *$variable* [=*default*]

Let's modify the previous example so that the golfer's score is a keyword parameter that defaults to "even" if it's not specified:

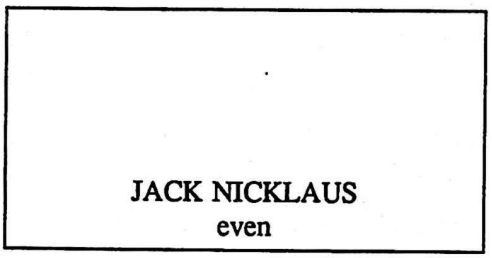
```
# type 2-line lower third
# score defaults to "even"
keyword $score = "even"
<Erase-Text>
<Sel-1>; type $1; <Newline>
<Sel-2>; type $score; <Center> 1
```

Do `golf "JACK NICKLAUS" score="+2"` **Enter**

creates the same display as before, while

Do `golf "JACK NICKLAUS"`
Enter

displays:



Flag Parameters are treated by K as a type of keyword parameter. A flag parameter is a keyword parameter that has only two possible values, 0 (false) or 1 (true). As with Do commands, use a leading hyphen for flag parameters:

```
-flag       # sets the flag to 1
-noflag     # sets the flag to 0
```

Let's add a flag parameter `-up` to the `golf` command, that forces the text into the upper left corner, instead of a lower third:

```
# type 2-line lower third
# or use -up for upper left
keyword $score = "even"
keyword $up = 0
<Erase-Text>
<Sel-1>; type $1; <Newline>
<Sel-2>; type $score
```

```
if $up    # upper left
  <Centr> t
  <Centr> e
else      # lower third
  <Centr> l
end
```

To display the score in the upper left corner, type

Do `golf "JACK NICKLAUS" -up` **Enter**

Returning from a K Subroutine

The `return` command forces an immediate termination of a K program:

`return` [*value*]

return can be combined with `if` to create a conditional termination:

```
if $key==<Compose>
  return
else
  ...
end
```

If a K subroutine is executing, `return` returns control to the calling K program. Additionally, the K subroutine may return a value to the calling program, which the calling program can assign to a variable. For example, this K program called `avg` takes a number of golf strokes and a number of holes, and computes the average number of strokes per hole:

```
        # compute strokes per hole
        # format: avg holes strokes
$holes = 1.0 * $1 # convert to real
return $2/$holes  # return the average
```

We could then call this K program from another K program, and assign the result to a variable:

```
$average = avg 18 72
```

Exiting from a K Program

Another way to return from a K program or subroutine is to `exit` from it. The `exit` command forces all K programs to terminate.

If exit is used in a subroutine, control is not returned to the calling program. All K programs cease executing, and the character generator returns to Compose mode.

The `exit_option` command determines which program conditions cause a K program to terminate:

`exit_option normal`
Terminates execution of a K program for these reasons:

- a system error message
- **Quit** is pressed
- `exit` command in the program
- normal program `end`

`exit_option cancel`
Terminates for all of the above conditions except a system error message.

`exit_option exit`
Terminates when the `exit` command is encountered. This is how you make a K program run forever; it cannot be canceled from the keyboard **Quit**.

`exit_option program`
Terminates when the the K program ends normally.

Data Type Conversions

There are times that a value of one type must be converted to a different data type to be used in a particular command. The following commands convert between data types:

[*$num*=] `asc2num` *string*
Convert a string to an integer or real number, and return the number. Example:

```
$a= "123"      # $a is a string
b= asc2num $a  # $b is integer 123
```

[*$str=*] `num2asc` *number*

Convert a real or integer number into a character string, return the string. Example:

```
$a= 123.45          # $a is a real
$b= num2asc $a      # $b is string
                    # "123.45"
```

[*$int=*] `str2num` *string*

Convert a text string of up to 4 characters into the (concatenated) ASCII values of that string. Example:

```
$a= "ab"            # $a is a string
$b= str2num $a      # $b is an
                    # integer
type $b             # types: 24930
```

[*$str=*] `num2str` *number*

Convert up to 4 bytes of binary data into an ASCII text string. Example:

```
$a= 6382179         # $a is an integer
$b= num2str $a      # $b is a string
type $b             # types: abc
```

[*$int=*] `integer` *number*

Convert real to integer by truncation, return integer value. Example:

```
$a = integer 3.9
type $a         # types: 3
```

String Operators

The following commands operate on character strings. K defines "character position 1" as the first character in the string.

[*$str=*] `substr` *string position* [*length*]

Return substring of specified string. If *length* is not specified, return remainder of string. Examples:

```
$a = "abcde"
$b = substr $a 3 2
type $b              # types: cd
$c = substr $a 3
type $c              # types: cde
```

[*$int*=] `index` *string1 string2*

Return starting position of the first occurrence of *string2* in *string1*. Return 0 if *string2* does not occur in *string1*. Examples:

```
$a = "abcde"
$a = index $a "bc"
type $a              # types: 2
$a = index $a "xyz"
type $a              # types: 0
```

[*$int*=] `length` *string*

Return the number of characters in a string. Example:

```
$a = length "abcde"
type $a              # types: 5
```

Cursor Ouput Commands

Here are the K commands that control the cursor output:

`cursor` *row column*
Position the cursor at the specified row (1-24) and column (1-64). Row 0 indicates the current row, and column 0 indicates the current column. For example,

```
cursor 2 3          # row 2 column 3
cursor 0 3          # column 3 of
                    # current row
```

`newline`
Position the cursor at the start of the next row.

`cursor_on`
Turn the cursor on.

`cursor_off`
Turn the cursor off.

`print` *param1* [*param2...*]
Display the specified parameters, separated by spaces. K variables may be used as parameters. For string and keystroke variables, `print` types the string or keystroke. For integers and real variables, `print` converts the number to a string and types the string. Examples:

K PROGRAM	DISPLAYS
`print "hello"`	hello

`$a="one"`
`$b=2345`
`print $a $b`
	one 2345

`$a=22`
`$b="catch"`
`$c=$b%%$a`
`print $c`
	catch22

`message` *param1* [*param2...*]
Display the specified parameters, clearing the remainder of the message line:

`message` uses the same K variable conversion rules as `print`.

`center` *param1* [*param2...*]
Display the specified parameters in the center of the current row of the monitor.

`clear_line` [*row*]
Clear the indicated row of the monitor. If no row is specified, clear the current row,

`clear`
Erase the monitor.

`status` *number*
Suppress output, until K playback ends, or another `status` command is encountered. Use `status` to increase the playback speed of some K programs.

Values for *number*:

0 = Normal, display all status updates and K prompts.
1 = Turn off status updates, but display K prompts.
2 = Turn off status updates and K prompts.

Control the Keyboard and K Playback

The following commands control interactions with the operator during the execution of a K program. K playback can be suspended at any time, to allow you to respond to a question, trigger an effect, etc.

Some of the following K commands return a value, which may be an integer, real, string or keystroke. The returned value may be assigned to a K variable. We illustrate these commands like this:

COMMAND	RETURNS	
[*$str=*]	*command*	string
[*$key=*]	*command*	keystroke
[*$int=*]	*command*	integer
[*$num=*]	*command*	integer or real

The playback control commands are:

[*$key*=] `getkey`
Wait for a key to be pressed, return its value. Examples:

`getkey`
waits for a single key to be pressed. After you press any key, the K program continues.

`$a=getkey`
waits for a key to be pressed, and assigns it to variable `$a`.

`type` *param1* [*param2...*]
Type the specified parameters, as if an operator had typed those keys. `type` follows the same K variable conversion rules as `print` above.

The following example uses `getkey` and `type` to provide verify protection on `<Erase-Text>`.

```
# allow normal 20-K operation
# but verify erase text

loop                # loop forever
  $key=getkey       # get a key
  if $key!=<Erase-Text>  # is it erase?
    type $key       # no, type it
    continue        # go to top of
    end             # loop
    message "Press y if you mean it"
    $verify=getkey
              # get verify key
    if  $verify == y  # was it a "y"?
      <Erase-Text>
                # yes, erase
    else            # no
      message "screen not erased"
    end             # end of if
  end               # end of loop
```

[*$num*=] `getnum` [*length initval*]
Prompt operator for a number, and return that value. The prompt appears on the status monitor, at the current cursor location. The operator enters a number just like filling in a menu entry, then presses **Enter**. Input is restricted to the digits 0-9, +, - and decimal point. The optional parameter *length* specifies maximum number of characters in the input field. The input field is initialized to *initval*.

[*$str*=] `getstring` [*length initval*]
Prompt operator for a string, and return that value. The operator enters the string, then presses **Enter**. The optional parameter *length* specifies the maximum number of characters in the input field. The input string is initialized to *initval*.

`user`
Suspend the currently executing K program and return control to the keyboard. The K program resumes execution when the operator presses **End-Input**.

`wait` *seconds* [*fields*]
Suspend execution of the currently executing K program for the specified amount of time. Examples:

```
wait 2      # wait for two seconds
wait 0 30   # wait for thirty fields
            # (1/2 second)
```

HINT: Use `wait 0` to force the monitor display to update.

`trigger` *number*
Wait for the specified GPI trigger to be closed.

`user_func`
Return control to the user until the current function completes. Typically, `user_func` is used during a playback effect, to allow the operator to start the effect manually, or to change speeds on the fly:

```
<Effect> w # program for scan wipe
user_func  # let user select speed
```

`user_func` can also be used to allow an operator to fill in a menu. Example:

```
<Font> e   # enter font edge menu
user_func  # let user fill it in
```

The operator moves the cursor up and down and fills in the menu entries. When the operator presses **Enter**, control returns to the K program.

`k_func`
Continue executing K program during a real-time effect. Normally, K programs stop while an effect is in progress, so that a playback speed can be entered from the keyboard. This is not always desirable. In the following example, K starts playing `myanimation` at speed 2, then switches to speed 8 after 3 seconds.

```
<Effct> a myanimation <Enter>

  k_func; 2  # start at speed 2
  wait 3; 8  # after 3 secs, speed 8
  user_func  # wait for animation to
             # end
```

Serial Input and Output

The following K programs control input and output through four serial ports. See "Serial Ports" in the User's Guide.

`portuse port=`
Return the `usage` of the specified port.

`getport use=`
Return the port number that has been configured for the requested `use`. Valid applications are listed in the command `portset`, under the `usage` parameter. They include `kbd`, `host`, `emem`, etc.

`input port= [length=] [terminator=]`
Receive data from the specified port, and return its value as a string. `input port` stops taking input when `length` bytes or a `terminator` character is received. `length` defaults to 1. If `terminator` is not specified, the command waits until exactly `length` bytes are received.

`inputbyte port= [-wait]`
Receive one byte of raw data from the specified `port`. If the `wait` switch is set, `inputbyte` will wait for input.

`output port= list=`
Send the specified parameters as ASCII strings via the specified port. It uses the same K variable conversion rules as `print` and `type`.

`outputbyte port= byte=`
Send one `byte` of raw data to the specified `port`.